INTEGRATION FRAMEWORK FOR USING PATTERNS IN MODEL DRIVEN DEVELOPMENT

Ruben PICEK

Author: ***Ruben Picek***
Workplace: ***Faculty of Organization and Informatics, Varaždin, University of Zagreb,***
E-mail: ***ruben.picek@foi.hr***

Abstract

Awareness of software's importance in today's environment each day is growing. As new types of applications appears, need for modern, high quality methodological ways of their development is rising. Last few years researches in software industry move in different directions. This doctoral dissertation is based on currently most skeptical software development paradigm called Model Driven Development (MDD). The idea is examine the possibilities of using patterns in context of MDD paradigm. Because in the methodological approach of software development is necessary to use some kind of development process, intention is discover applicability of existing development process for MDD paradigm. Then, in the context of patterns it is necessary to define what kind of patterns is possible to use, and in what kind of form patterns has to be defined to use them for MDD. Furthermore, when this criteria's are satisfied, using patterns in context of MDD paradigm to provide automated SW development –what that paradigm promote, it is necessary to define framework for using patterns in MDD paradigm and then integrated its with modern software development methodology suitable for model driven development.

AN OVERVIEW OF SOFTWARE ENGINEERING

In software development during 90's two paradigms were dominated. In the early 90's it was popular the concept of Computer Aided Software Engineering based on CASE tools and 4GL while in the second half of the decade strong influence in software industry take object oriented (OO) paradigm. Although it wasn't fulfilled all expectation, OO paradigm preserve to nowadays and during this years becomes a foundation for: component based development (CBD), OO languages, modeling notation today known as UML and philosophy called round trip engineering. Today's situation in software industry pointing that scenario repeats. Although use of OO development solves many problems in today's SW industry, evolution in all segments of society causes new challenges. Requirements of new and/or existing systems are growing, systems are complex and it is hard to build them on time and on budget. As an answer to these challenges, a wide spectrum of new approaches occurred, varying from buzzwords to comprehensive methodologies. The most important are: Service Oriented Architecture (SOA), Model Driven Development (MDD), Agile Development, Software Product Lines (SPL), Model Driven Architecture (MDA) and Software Factories (SF).

Actually, current state in SW industry can be called paradigm shift like it was in 90's, when development shifts from structured to object oriented. Today's modern development is directed toward new paradigm called model driven development. Currently paradigm have high position in software industry. The idea of MDD paradigm is using models, as primary artifacts, not only for documenting software but also for transforming into programming code with a given level of automation. Some of the above buzzwords are concrete realization of this paradigm. Paradigm combines techniques like domain analysis, meta modeling, generation based on the models and use of modeling languages. All this shows how much industry all this years growth despite problems and new challenges which it were surrounded. Although in academic community the term is very well established, concrete realization in industry encounter on problems and it is not possible to predict evolution of MDD in future. One of the biggest problem is automated transformations development. Many organization which develop applications will have to face it with challenge of developing software using MDD paradigm to decide will they go this way or not. Currently this organization are in the initial phases of adopting this paradigm with a big amount of skepticism.

Beside all this, in SW industry are present myths related to MDD paradigm. Some of them have long history while the roots of others can be traced from last paradigms (origin in the 80's and 90's and related to experiences with CASE tools) or they are not actually myths but the reality which is now being transposed onto model-driven approaches in general.

Most frequently mention myths are [4]:

- Model always equals UML model.
- Modeling always implies having all specifications in the form of visual models, at the same level of abstraction as the implementation source code.
- Code generated from a model is ugly and not really suitable for human readers.
- Generated code has lower performance than handcrafted code.
- A generative approach ties you too strongly to a tool and a specific set of technologies
- Model-driven approaches are incompatible with the concept of agility.
- Model-driven is the same as CASE and we know it doesn't work.
- Regeneration is not practical and model and code will always get out of synch at some point.
- It takes longer to write the generator than to hand-code everything.
- Adapting the generator to accommodate changes takes longer than manually changing the generated code.
- Code generators are not object-oriented and code produced by code generators is not object-oriented. It is always better to write an object-oriented framework than to write a code generator. In fact, writing a code generator is a cop-out from good framework design.

Parallel with OO paradigm, patterns have been successfully used in SW industry as solution for recurrences problems which often appears in some context. When we today use term pattern usually we think on design patterns, but in last few years patterns by their definition and number evolve. So this new situations has to be examine. Furthermore when today's software development process (heavily and light) is analyzed it can be notice that their relation with model driven development is not defined. SW industry as well observe software development from economic aspects and more and more all types of artefact are view as reusable assets.

All aspects mention above, point out that are many question are still opened and it is worth to examine some of them.

MODEL DRIVEN DEVELOPMENT PARADIGM

As already mention, in the last few years software development has been faced with many challenges and as an answer to these challenges, a wide spectrum of new approaches occurred, varying from buzzwords to comprehensive methodologies. One of the most prominent paradigms is Model Driven Development (MDD). MDD represents a set of approaches, theories and methodological frameworks for industrialized software development, based on the systematic use of models as primary artifacts throughout the software development cycle. It targets two roots of software crisis – *complexity* and *ability to change*.

First of all, let us start with definition of MDD paradigm. The basic idea of this paradigm is to move the development efforts from programming to the higher level of abstraction, by using models as primary artifacts and by transforming models into source code or other artifacts The ultimate objective is the automated development (fully or partly). According to [17], MDD is a style of software development where models are primary software artifacts. Other artifacts and code are generated from them, according to best practices. In [12], MDD is defined as software engineering approach consisting of the application of models and model technology to raise the level of abstraction at which developers create and evolve software, with the goal of both simplifying and formalizing the various activities and tasks that comprise the software life cycle. According to [5], MDD refers to a set of approaches in which code is automatically or semi automatically generated from more abstract models, and which employs standard specification languages for describing those models and the transformations between them.

These definitions make clear that the focus of the MDD is a shift from the programming to the modeling. Models are the key artifacts. Currently, models are mostly used as sketches that informally convey some aspects of a system or they can be used as blueprints to describe a detailed design that is then manually implemented [17]. Use of models as documentation and specification is valuable, but it requires strict discipline to ensure that models are kept up to date as implementation progresses. Time constraints mainly cause that the initial models are not updated during the design and implementation, and inaccurate models are harmful. In MDD, models are used not just as sketches or blueprints, but as primary artifacts from which efficient implementations are generated, transforming models into programming code or other artifacts characteristic to domain.

As described above, MDD shifts the emphasis of application development away from the platform, enabling developers to design applications independent of the platform-level concerns. Platform is the province of developers with the platform specific expertise. Platform expertise is involved as late as possible, it means at transformations, rather than, being rediscovered many times during a project. Likewise, decisions about the implementation architecture are directly encoded in the transformation engine rather than documented as architectural decisions [17].

According to Selic [14], the essence of model driven development is about two things. One is *abstraction*, in terms of how we think about the problem and then how we specify our solutions. Second thing that often gets forgotten is the introduction of more and more *automation* into the software development, specifically by using computer based tools and integrated environments.

MDD-style should enable automation to go much further. A software development project needs to produce many non-code artifacts; some of these are completely or partially derivable from models. The following list gives some common examples of artifacts that are generated from models, but you can probably think of others [17]: documentation, test artifacts, build and deployment scripts, other models, pattern application.

According to the above mentioned definitions, the heart of MDD paradigm makes: models, modeling and model transformation. Modeling and models are the central point of contemporary software development. But, one fact related to models has to be emphasized. There is a big difference of *what models represent* and *how there are used*. Traditional models are just sketches, and blueprints for design. In order to be suitable for the MDD, models must satisfy additional criteria – they must be machine readable. Machine-readability of models is a prerequisite for being able to generate artifacts. There are two types of transformations: model to model (M2M) and model to code (M2C). Automated model transformations are the key for realization of the MDD idea.

Two leading realizations of MDD paradigm are: The Object Managements Group (OMG) approach called Model Driven Architecture (MDA) and Microsoft's Software Factories (SF). According to [12] it is too early to predict which, if any, of the current MDD approaches will be accepted as an industrial standard.

BENEFITS OF THE MDD

According to [17], [15], MDD has the potential to greatly improve current practices in software development. This potential manifests in overcoming the current challenges – reducing the cost of development and increasing the consistency and quality of software. Some advantages of an MDD paradigm are [17], [15]: increased developer productivity, maintainability, reuse of legacy, adaptability, consistency, repeatability, improved stakeholder communication, improved design communication, capture of domain knowledge, models as long-term assets and ability to delay technology decisions.

CURRENT PROBLEMS

The primary goal in MDD paradigm is to raise the level of abstraction at which developers operate. It should reduce both the amount of developer's efforts and the complexity of the software artifacts that the developers use [12], [11]. Of course, there is always a trade-off between simplification by raising the level of abstraction and oversimplification, where details for any useful transformation are missing. As you can assume, problems are bound to model abstractions at different stages of the software life cycle. The open issue is *how to transform a model at one level of abstraction, into a model or code at a lower level?* In trying to answer this question, new ones arise.

How to use models? Some developers use models only for sketching, others for blueprinting while MDD community presumed models as programming language.

Which notation and modeling language should be used in order to provide automation? The standardization of modeling notations is unquestionably an important step for achieving MDD. Standardization provides developers with uniform modeling notations for a wide range of modeling activities. In SW industry today, the Unified Modeling Language (UML) is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems. The UML represents a collection of best engineering practices which have been proven in the modeling of large and complex systems. Although UML is widely recognized and used as modeling standard, it provoked a lot of criticism.

Is UML suitable as model programming language? The notion of UML 2.0 as a model programming language is predicated on the belief that the use of higher levels of abstraction will make developers more productive than current programming languages. Fowler [8], discusses whether this opinion is true. He doesn't believe that graphical programming will succeed just because it is graphical. Indeed he has seen several graphical programming environments that failed - primarily because it was slower to use them than writing code. (Compare coding an algorithm to drawing a flow chart for it.). Even if UML is more

productive than programming languages, it will take the time to become accepted. Most of the people for many reasons are not using programming language they consider to be the most productive. Furthermore Greenfield et al. [10] argue that although UML 2.0 is a useful modeling language, it is not an appropriate language for MDD, because UML is designed for documenting and not for programming. They promote use of special-purpose, domain-specific languages (DSL's). Clearly, UML or any other MDD language faces significant hurdles to demonstrate sufficient value to satisfy the needs of all the different kinds of MDD users.

According to [12], MDD creates other problems, like: redundancy, rampant round-trip problems, moving complexity rather than reducing it and more expertise that is required.

SOFTWARE PATTERNS

Patterns in software development have short history. When in the beginning of 90's object oriented paradigm appears, patterns has become a interesting area because they become a way of realization object oriented main characteristics – reusability.

Term pattern can evoke a variety of interpretations and definitions mostly depending on context of use. According to different authors [16], [7], [13], [15] *pattern* is a best practice solution to a recurring problem for a given context. But each solution, algorithm or best practice isn't pattern. Main characteristic of patterns is ability to use it all over again in recurring problems (recurrence). Pattern encapsulate a designer's time, skills and knowledge to solve a software problem and include all essential factors that pattern could be reused in similar situations. Recurrence is not only pattern characteristics. Pattern must be suitable (fit) for the problem and useful in given context. When *recurrence* is quantitative characteristic which is shown by context and problem definition, *suitability* and *usefulness* are qualitative characteristic where suitability clarify *how* the pattern will contribute to the problem solution, while usefulness explains *why* pattern will be useful.

Definition of software pattern used in this dissertation is: *Software pattern is a recurring structural concept used in context of software development which consists of problem definition and best practice solution and can be applied on all levels of abstraction during software development lifecycle.*

In the context of software development, software pattern is reusable concept on higher level of abstraction then lines of programming code, individual classes and components. Patterns provide a very powerful way of improving today's software development by identifying best practices and design expertise that can be captured in tools and then available for reuse. Main advantages of using patterns in software development are [1]: improve productivity, reduce development time, minimize complexity, increase quality, improve governance, gain business agility, leverage IT skills, promote open standards, close the gap between business and IT, improve cost. Two more advantages according to [2] are: improving managing of developing projects and simplicity of software architecture.

PATTERN CLASSIFICATION AND STANDARDIZATION

Since beginning of their use in software industry, development, studying and use of patterns have increasing trend. Their prominent use, impact to paying more intention to pattern development and the result was quite number of new patterns in all segments of software development. Each organization developed patterns organize in catalogs and place them in own pattern repository. Different organizations have patterns which were suitable for their way of work and used technology. Direct consequence of that is big number of different structured patterns and absence of unified classification. On the basis of examine literature

and new knowledge aim is make a fundamental classification of today known patterns and propose new criteria for pattern classification.

Number of patterns exponentially grows, so not only the unified classification which will contribute to quality development, faster findings and use is needed but also is needed a initiatives for pattern standardization. Software industry itself evolves, so the model driven approaches with idea of using models for automated development have bigger importance. To include patterns in MDD scenario it is necessarily define a standardize way of patterns organization, specification, implementation and packing.

Towards increasing the return on investment it is introduced term *reusable software asset* which includes all types of different artifacts used in development process. Furthermore to ensure their reusability and standardization, Reusable Asset Specification (RAS) was developed for providing a standardize way of *software asset* archiving, searching, organizing, specifying, implementing, packing and sharing. In dissertation it will be explore how patterns can be observe as *software asset* and how to implement them using RAS standard.

CONDITIONS FOR USING PATTERNS IN MDD

Using patterns in MDD context also arises some questions: *Can advantages that pattern brings to development be identified and used in MDD realization? How to define patterns adequate for using in MDD environment*?

Closer look at patterns advantages and advantages which emphasise MDD paradigm easily conclude that patterns can significantly improve realization of MDD paradigm. But new question arises - How? Complementary relationship between patterns and MDD can be seen in two forms:

- *Software patterns provide content for MDD.* Expertise captured in patterns represents best practice for the problem solution with opportunity for reuse and that is the main reason why patterns are desirable content for MDD.
- *Model automation - as based idea, can be achieve with using patterns.* Traditionally, patterns are written down as documents, often with the aid of UML models to explain the pattern and then applied manually. But if pattern is not define in the form of text which programmer has to manually implement but is packed as reusable asset with encapsulated implementation which also represent best practice, pattern can be automated from conceptual level to programming code.

Results of using patterns in MDD can be visible in: reducing time to react, enabling on demand design and development, reducing complexity and increasing productivity and quality of software [9].

According to all this, idea in dissertation is defining conditions for using patterns in MDD paradigm which has to be satisfied regardless to existing patterns or those which will has to be developed in future.

MDD METHODOLOGIES ANALYSIS AND INTEGRATION FRAMEWORK DEVELOPMENT

Despite its merits it is strange how (both) MDD realizations remains insufficient for software development, in the sense that it does not provide a concrete and comprehensive process for governing software development activities [6]. My aim is not to suggest that industry has to have a new methodology for MDD but only to emphasize the fact that there is still a lack of formal process for model driven development. Therefore it will be examine suitability today's modern software development methodology for MDD development. After

that, it will be defined framework for using patterns in MDD development. In order that framework become methodologically acceptable and practical it is necessary that his design includes all segments of pattern lifecycle. Pattern lifecycle will be developed on the base of asset lifecycle, because one of the condition must be that pattern has to be defined as reusable asset. Realizations of each identifying segments will be then methodologically described. Examination of this dissertation will be finished with integration developed framework with modern methodology which appears potentially adequate for MDD development.

CONCLUSION

The "modest" intent of MDD is to improve software quality, reduce complexity, and improve reuse by enabling developers to work at reasonably higher levels of abstraction and to ignore "unnecessary" details. In practice, however, MDD raises a number of significant issues which are still open.

In this dissertation it will be examine only one aspect in MDD paradigm. Main thread is investigating possibilities of using patterns in MDD paradigm context. Primarily focus is govern to:

- Developing methodological framework which will help developer in using patterns in MDD software development projects and
- Integrating developed framework with adequate modern methodologies

References

[1] *** IBM developerWorks: Pattern Solution,, 2007, http://www-128.ibm.com/developerworks/rational/products/patternsolutions/

[2] ACKERMAN, L., GONZALEZ, C. *The Value of Pattern Implementations*. DrDobb's Portal, http://www.ddj.com/cpp/199204017, 2007.

[3] APPLETON, B. *Patterns and Sotfware: Essential Concepts and Terminology*, http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html#PatternDefinition, 2000.

[4] BETTIN, J. Model-Driven Software Development, SoftMetaWare, <http://www.softmetaware.com/whitepapers.html> (pristupano: 15.12.2007.) , 2004

[5] BROWN, A. W., IYENGAR, S., JOHNSTON, S. A Rational approach to model-driven development, IBM System Yournal, Vol 45, No 3. 2006., p. 463-480., http://www.research.ibm.com/journal /sj/453/brown.html, (05.01.2007.)

[6] CHITFOROUSH, F., YAZDANDOOST, M., Ramsin, R. *Methodology Support for the Model Driven Architecture, Asia-Pacific Software Engineering Conference*. APSEC Volume, Issue, 4-7 Dec. 2007 Page(s): 454 – 461, 2007.

[7] ELSSAMADISY, A. *Patterns of Agile Practice Adoption*. InfoQ, 2006.

[8] FOWLER, M. *UML As Programming Language*. http://martinfowler.com/bliki/UmlAsProgrammingLanguage.html, (25.02.2007.)

[9] GARDNER, T., YUSUF, L. *Combine Patterns and Modeling to Implement Architecture-Driven Development*. 2006. http://www.ibm.com/developerworks/ibm/library/ar-mdd2/

[10] GREENFIELD, J., SHORT, K., COOK, S., KENT, S. *Software Factories - Assembling Application with Patterns, Models, Frameworks and Tools*. Wiley Publishing, 2004.

[11] GREENFIELD, J., SHORT, K. Moving to Software Factories, http://blogs.msdn.com/askburton/archive/2004/09/20/232065.aspx, (25.05.2007.), 2004.

[12] HAILPERN, B., TARR, P. *Model-driven development: The good, the bad, and the ugly*. IBM System Yournal, Vol 45, No 3. 2006., p. 451-461., http://www.research.ibm.com/journal/ sj/453/hailpern.html, (05.01.2007.)
[13] LARSEN, G. Model-Driven Development: Assets and Reuse, IBM System journal Vol 45, No 3, 2006., http://www.research.ibm.com/journal/sj/453/larsen.html
[14] PIERSON, H. ARCast #5, 2007. http://channel9. msdn.com/ Showpost.aspx?postid=132943, (09.02.2007.)
[15] SWITHINBANK, P., CHESSELL, M., GARDNER, T., GRIFFIN, C., MAN, J., WYLIE, H., YUSUF, L. *Patterns: Model-Driven Development Using IBM Rational Software Architect*. IBM Redbooks, 2005.
[16] YU, C. *Model-Driven and Pattern-Based Development Using Rational Software Architect, Part 1: Overview of the Model-Driven Development Paradigm With Patterns*. http://www-128.ibm.com/developerworks/rational/library/06/1121_yu/, 2007.
[17] YUSUF, L., CHESSEL, M., GARDNER,T. *Implement model-driven development to increase the business value of your IT system*. http://www-128.ibm.com/developerworks/library/ar-mdd1/, (14.04.2006.)